Kalman Filter

Shubhankar Kulkarni

October 14, 2022

Contents

Abstract

The need for filtering stems from the probabilistic nature of things making it a problem to be solved mathematically rather than just a binary answer to whether a certain commodity has a particular value. In context towards tracking systems, the probabilistic nature of any measurement mandates the need for a quantitative mechanism that can model the distribution of the probability of the object to be tracked. This mechanism can then be used to predict the future values of that object.

1 Introduction

The idea of fitting a model to describe a tracking problem is almost always not practical since the object in consideration does not have a defined path to travel due to which tracking is employed in the first place. A change in model upon some old model metrics getting violated is also not very optimal since it means that the object keeps on continually diverging from the older model.

This makes it necessary to update the model fit after every measurement reading to avoid the undesirable tracking behavior. This is the fundamental on which filtering is based upon. A state space model needs to be formed out of the system being tracked for applying the filter. Every estimation filter can be roughly said to encompass two main steps in a perpetual cycle- prediction of the next state from current data and updation of this prediction by weighing it with the measurement reading at that instance.

In the following sections, we will explore the **Kalman filter** which is one of the most popular mechanisms employed for estimation of linear tracking problems. Non-linear problems could be solved by usage of the more computationally intensive **Extended Kalman filter** or by linearizing them to be applied with Kalman filter.

2 Methodology

The Kalman filter follows the predict and update model of the filters and uses a term called **Kalman Gain** which is computed in every iteration to be added to the update of the state at next stage using the measurement data and the prediction from the previous state.

The uncertainties in the prediction and measurement are also factored in calculating the Kalman gain with the distribution of these uncertainties being quantified through their standard deviation. This leads to calculation of the Kalman gain using the variances of those distributions respectively. The Kalman filter primarily uses the matrix notations to define the noises, state transitions and measurement relations with the states. Given below is the general procedure of the filter-

Given that the system is converted into a model of state space equations. The *first step* is to calculate the prediction of the next stage based upon the current stage output. The equation can be given by-

$X_{t,t-1} = \Phi * X_{t-1,t-1}$ where Φ represents the state transition matrix.

Φ gives us the relation of how a state value changes with respect to it's predecing term. The first subscript represents the timestamp a value belongs to and the second subscript represents the timestamp of measurement taken for that state value. For the initial state value assumption, any arbitrary value can be passed on. The Kalman filter has a characteristic of converging on to the expected region from any initial value with the convergence time depending upon the value itself.

The *second step* is to predict the next state's covariance matrix which can be represented by-

$S_{t,t-1} = \Phi S_{t-1,t-1}\Phi^T + Q$ where S represents the state covariance matrix & Q matrix represents the dynamic noise, a quantitative estimate of the prediction's accuracy.

Every state updation we do is based upon some assumption and that certainty is captured within the X matrix but the real world is seldom like that. $S_{t,t-1}$ captures the uncertainties for the model which can be translated from the Gaussian distribution as variances of a state and covariances for a couple of states which denote how change in one state affects the other. The Kalman filter makes the uncertainties flow through the loop using the state covariance matrix and they contribute to the Kalman Gain value as well.

The *third step* is to measure the next timestamp's sensor measurement which can be represented by-

Y_t where Y_t is the measured data corresponding to the prediction timestamp.

The *forth step* is to calculate the *Kalman Gain* which can be represented by-

$K_t = S_{t,t-1}M^T[MS_{t,t-1}M^T + R]^{-1}$ where K_t represents the Kalman Gain, M represents the measurement matrix which links the states with the measurements & R matrix represents the measurement noise accounting for sensor's in-built uncertainty.

The *fifth step* is to compute the updated state value which is a combination of the prediction & the sensor measurement and is weighted by the *Kalman Gain* for that iteration. This can be represented by-

$X_{t,t} = X_{t,t-1} + K_t(Y_t - MX_{t,t-1})$ where $X_{t,t}$ represents the updated state for the next timestamp as seen by the subscripts.

The *sixth step* is to compute the updated state covariance matrix which is represented by the following relation-

$S_{t,t} = [I - K_tM]S_{t,t-1}$ where $S_{t,t}$ depends upon the *Kalman Gain* and the measurement matrix and the covariance matrix prediction from the second step. I is the identity matrix.

Computing the next state value and the associated state covariance matrix signifies that we have a mean value denoted by the $X_{t,t}$ state vector and the uncertainty around it denoted by the $S_{t,t}$.

The last step is to make the next timestamp as the previous one i.e. $t = t-1$. This would make the updated state vector and the state covariance matrix as the previous ones and we can continue the loop to calculate the next timestamp's state and state covariance matrices. This iterative process is the underlying algorithm of the **Kalman filter**.

2.1 Applying KF to a 1D Constant velocity problem

The problem statement here is that we are tasked with testimating the position of an object which is moving in a single dimension with a constant velocity. Using the above defined methodology, we create a state space equation of the system by assuming the states to be position x and velocity $xdot$ which can be shown as

$$x_{t,t} = x_{t,t-1} + T(xdot_{t,t-1}) \quad xdot_{t,t} = xdot_{t,t-1}$$

The measurement data from the sensor we get is of position and hence we can have measurement y as $y_t = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_{t,t} \\ xdot_{t,t} \end{bmatrix}$

Both the equations do not account for the noises that can accumulate within the predictions as well as sensor readings due to a system not behaving in constant velocity manner and sensor limitations in real life respectively.

The system noise can be assumed to be acceleration and is accounted as a Normal distribution in the velocity equation$N(0, \sigma^2_{xdot})$.
The measurement noise can be assumed to be a result of the hardware limitations and is added as a Normal distribution in the measurement and state relation equation as$N(0, \sigma^2_n)$.

The dynamic noise matrix can be created as $Q = \begin{bmatrix} 0 & 0 \\ 0 & \sigma^2_{xdot} \end{bmatrix}$. Here the position is assumed to have to standard deviation and hence zero variance making the covariance of velocity and position become zero as well.
The measurement noise matrix can be created as $R = \sigma^2_n$.

Since we have the measurement data as an input to us, we assume a time series with timestamps corresponding to each of the measurement value. Hence our loop would run as long as the index of measurement array returns a value. Additionally, we would assume the initial state covariance matrix $S_{t-1,t-1}$ to be an identity matrix of dimension $2 * 2$. The initial velocity value is assumed to be the $(y_2 - y_1)/t_1$. The initial position value is assumed to be $0.9 * (y_1)$. We also assume a certain value for the standard deviations of acceleration and measurement.

With this, we run the Kalman Filter algorithm and calculate the successive stage state values and state covariance matrices. Then as per the computed Kalman gain, we get the updated state and state covariance matrix values and use them in turn to calculate the successive readings. This iteration gives us an approximation of state values which we can then compare with the given measurement values to check the confidence of the filter towards the prediction and measurement.

2.2 Applying KF to a 2D Constant velocity problem

The problem statement here is that we are tasked with an ultra wide band sensor data used for tracking. The data contains the x-coordinates and y-coordinate measruements of the object being tracked. This extends our problem into the 2d space which means that both the x and y position become states. Assuming a constant velocity model, the repsective x and y velocities also become states which means that the toal state count is 4 and hence the state vector is $4 * 1$ matrix denoted by-

$$X = \begin{bmatrix} x_{x(t,t)} \\ xdot_{x(t,t)} \\ x_{y(t,t)} \\ xdot_{y(t,t)} \end{bmatrix}$$

The outputs of the system can be considered the x and y positions making the measurement matrix a $4 * 1$ matrix denoted as-

$Y = \begin{bmatrix} y_{x(t,t)} \\ y_{y(t,t)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * X$ which gives us the M matrix.

The state space equation of the model is formulated and we get a state transition matrix Φ of dimension $4 * 4$ given by $X_t = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} * X_{t-1}$

The 2 subscript notation is not needed at this point.

The system is assumed to be a constant 2D velocity model and hence practically would possess a system noise in terms of acceleration that is added up the respective velocity equation denoted by $N(0, \sigma^2_{xdot})$ & $N(0, \sigma^2_{ydot})$. The dynamic noise matrix Q thus becomes a $4*4$ matrix denoted as $Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & sigma^2_{xdot} & 0 & \# \\ 0 & 0 & 0 & 0 \\ 0 & \# & 0 & sigma^2_{ydot} \end{bmatrix}$

The # represents an arbitrary small value representing the covariance of both the velocities.

The measurement noise matrix R is a $2 * 1$ dimensional one with the values taken as the variance of the sensor readings $N(0, \sigma^2_{nx})$ & $N(0, \sigma^2_{ny})$.

We would assume the initial state covariance matrix $S_{t-1,t-1}$ to be an identity matrix of dimension $4 * 4$. The initial x and y velocities are assumed to be $(y_x 2 - y_x 1)/t_1 \& (y_y 2 - y_y 1)/t_1$ respectively. The initial positions are assumed to be $0.9 * (y_x 1) \& 0.9 * (y_y 1)$. We also assume a certain value for the standard deviations of accelerations in x and y and measurements of x position and y position.

This goes through the Kalman filter loop which predicts and updates the state values and state covariance matrix values ultimately giving us a estimate based upon a weighted confidence in the prediction and sensor reading.

3 Observations

3.1 1D constant velocity tracking

The variation of the noises in the system and that of the sensors produce a difference in the way the filter computes the updated values and hence this is something that needs to be determined as per the subjective scenario.

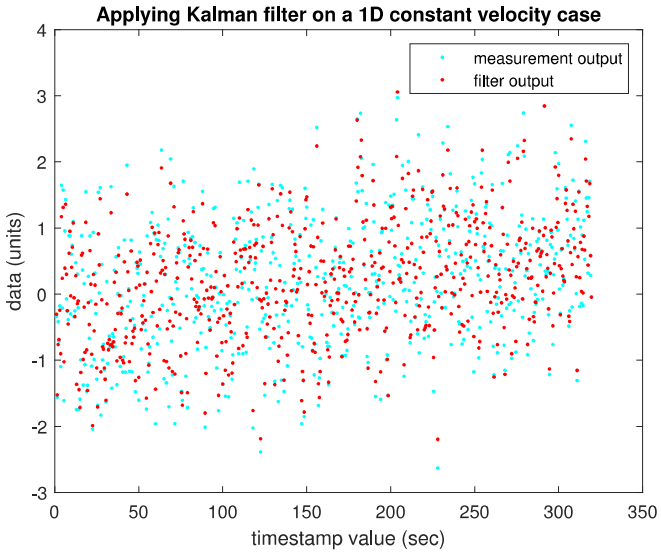

Figure 1: Filter output versus measurement data

For the above figure, the measurement noise R is kept at 0.01 and the dynamic noise Q is kept at $Q = \begin{bmatrix} 0 & 0 \\ 0 & \sigma_{xdot}^2 \end{bmatrix}$ with the $\sigma_{xdot} = 0.3$.
It can be seen that the updated state values from the filter are dispersed similar to the sensor readings but still are less spread out from each other indicating a stronger trend than the measurement data. This means that the filter takes a weighted consideration of both the measurement and the prediction.

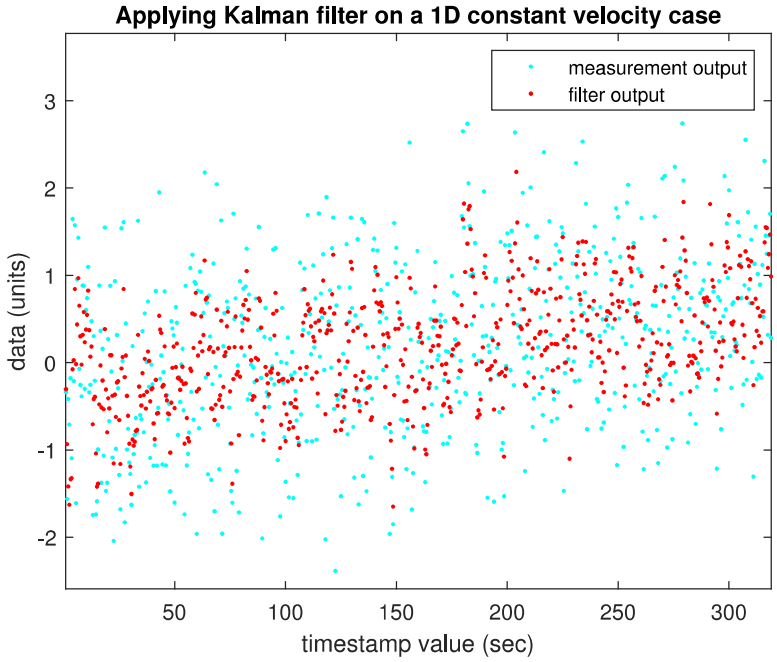

Figure 2: Filter output versus measurement data with more measurement noise

For the above figure, the measurement noise R is increased to 0.75 and the dynamic noise Q is kept same.
It can be seen that the updated state values from the filter are condensed in a region indicating a clear trend while the measurement data is dispersed sparsely in comparison. This means that the filter outputted values that subsided with the prediction value to a larger extent than the measurement data.

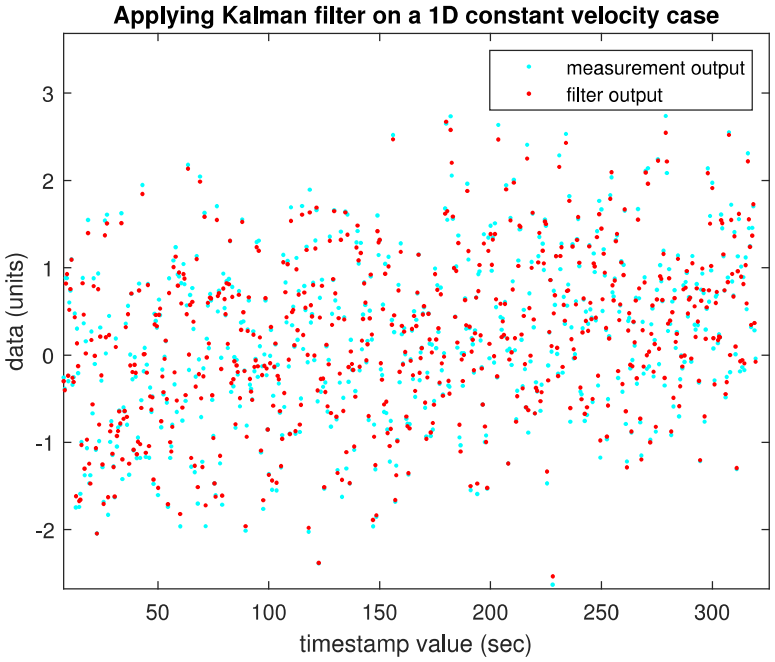

Figure 3: Filter output versus measurement data with more dynamic noise

For the above figure, the measurement noise R is decreased back to 0.01 and the dynamic noise is increased to $Q = \begin{bmatrix} 0 & 0 \\ 0 & \sigma_{xdot}^2 * 7.5 \end{bmatrix}$ with the $\sigma_{xdot} = 0.3$. It can be seen that the updated state values from the filter are condensed in a region indicating a clear trend while the measurement data is dispersed sparsely in comparison. This means that the filter outputted values that subsided with the prediction value to a larger extent than the measurement data.

3.2 2D constant velocity tracking

The variation of the noises in the system and that of the sensors produce a difference in the way the filter computes the updated values and hence this is something that needs to be determined as per the subjective scenario.

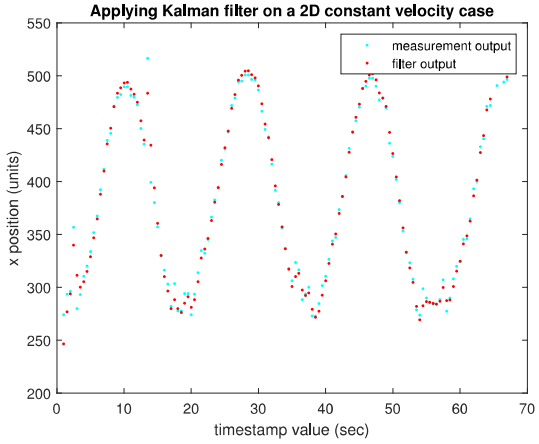

Figure 4: Filter output versus measurement data

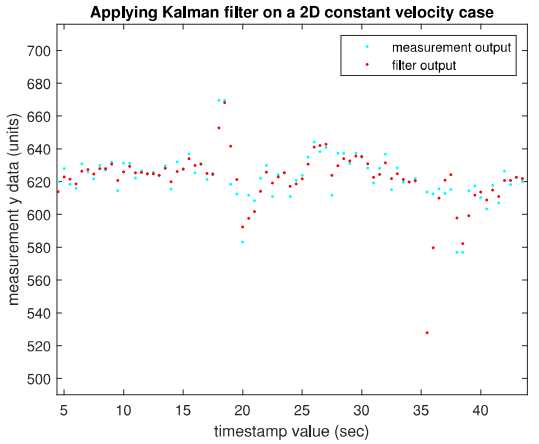

Figure 5: Filter output versus measurement data

For the above figure, the measurement noise is kept at $R = \begin{bmatrix} 1 & 0.01 \\ 0.01 & 1 \end{bmatrix}$ and

the dynamic noise is kept at $Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & sigma^2_{xdot} & 0 & \# \\ 0 & 0 & 0 & 0 \\ 0 & \# & 0 & sigma^2_{ydot} \end{bmatrix}$
with the $\sigma_{xdot} = 0.5$, $\sigma_{ydot} = 0.5$ and $\# = 0.1$.
It can be seen that the sensor readings are a set of sinusoids with the x measurements having a higher amplitude and the y having a less prominent share with a lower amplitude. For the symmetric noise values in both x and y for both the dynamic noise and measurement noise matrices, the x & y position of the filter output follow the trends of the measurement x and y data indicating a higher fidelity towards the measurement than the prediction but they aren't totally identical to the measurement data values.

For an increased Q/R ratio by reducing R and keeping Q unchanged.

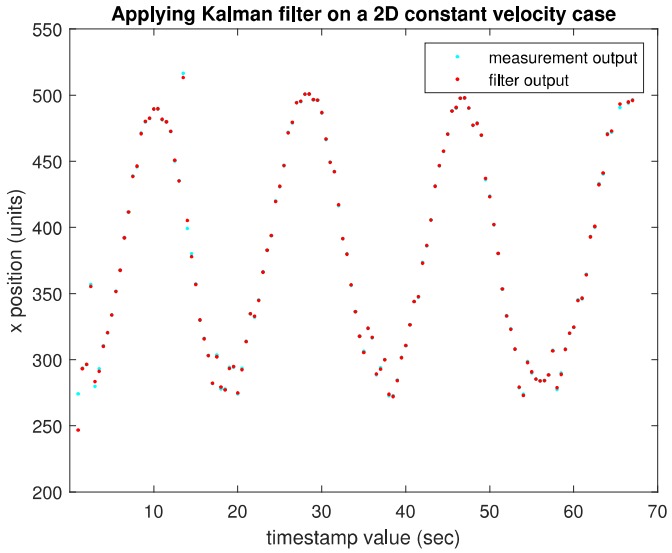

Figure 6: Filter output v/s measurement data with less measurement noise: x

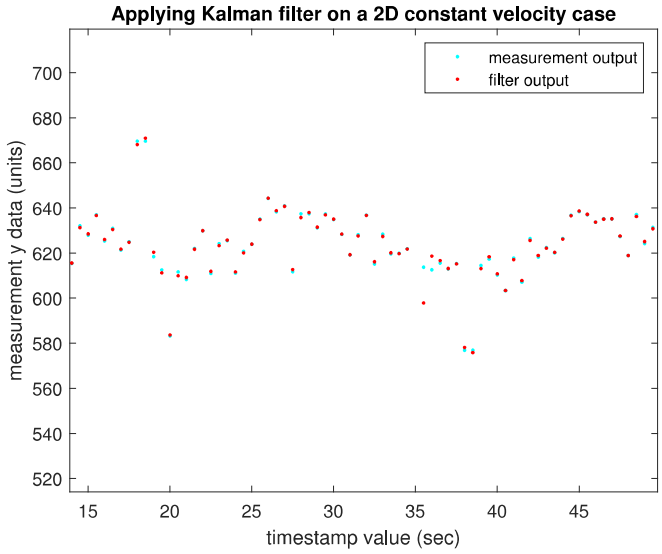

Figure 7: Filter output v/s measurement data with less measurement noise: y

For the above figure, the measurement noise is decreased$R = \begin{bmatrix} 0.1 & 0.001 \\ 0.001 & 0.1 \end{bmatrix}$ and the dynamic noise is kept unchanged.
It can be seen that the updated state values from the filter are almost coinciding with the measurement x & y data indicating an even increased confidence in the measurement. This behavior is resultant of an increased Q/R ratio.

4 Results and conclusion

The conclusion that can be drawn from the above plots is that there is no true value of any quantity that can be measured. Any estimate is always a weighted component of the sensor reading and the prediction made by the filter. The updated output tends to follow the trends of estimate values for lower ratios of Q/R and the trends of the measurement are followed more for the higher values of Q/R.

5 Appendix

5.1 Code: 1D Constant Velocity KF

```
clc;
clear;
urlwrite("https://cecas.clemson.edu/ahoover/ece854/labs/1D-data.txt","measurement_Q1")
%%
inp_filename = "measurement_Q1.txt"
fid = fopen(inp_filename); % open the file
iCtr = 0;
y_q1=[];
while ~feof(fid) % loop over the following until the end of the file is reached.
  line = fgets(fid); % read in one line
  y_q1(end+1) = str2double(line);
end
%%
y_q1=y_q1';
%%

%assuming a time array corresponding with measurement data with a frequency of 1Hz
time_q1=(1:0.5:length(y_q1)/2);
%%

%assuming a position array corresponding with a predicted state in column 1
%and an updated state in column 2
x_q1=zeros(length(y_q1),2);

x_q1(1,1:end) = y_q1(1)*0.9;

%%
%assuming a velocity array corresponding with each position state for a
%constant velocity model
x_dot_q1 = zeros(length(y_q1),2);

x_dot_q1(1,1:end) = y_q1(2)-y_q1(1);
%%
%we know that there's measurement noises and dynamic noises
S_prev_q1 = [0.02, 0.1; 0.04, 0.25];
S_next_q1 = 0;
R_q1 = 0.01;
M_q1 = [1, 0];
var_x_dot = 0.3;
Q_q1 = [0, 0; 0, var_x_dot^2];
%%
%Kalman_filter(x_q1,x_dot_q1,y_q1,time_q1,S_prev_q1,S_next_q1,R_q1,M_q1,Q_q1,2)
```

```
%creating the loop for running the filter
for i=2:1:length(y_q1)-1
    %step 1: Predict next state
    phi_q1 = [1, time_q1(i)-time_q1(i-1); 0, 1];
    X_pred_q1 = phi_q1*([x_q1(i-1,2),x_dot_q1(i-1,2)]');
    %placing predictions in the state variable array
    x_q1(i,1)=X_pred_q1(1);
    x_dot_q1(i,1)=X_pred_q1(2);

    %step 2: predict next state covariance
    S_next_q1 = phi_q1*S_prev_q1*(phi_q1') + Q_q1;

    %step 3: obtain measurements (there as input)
    Y_q1 = [y_q1(i);0];

    %step 4: calculate Kalman Gain
    K_q1 = S_next_q1*(M_q1)'*((M_q1*S_next_q1*(M_q1)'+ R_q1)^-1);

    %step 5: update state
    X_upd_q1 = X_pred_q1 + K_q1*(y_q1(i)-(M_q1*X_pred_q1));
    %placing predictions in the state variable array
    x_q1(i,2)=X_upd_q1(1);
    x_dot_q1(i,2)=X_upd_q1(2);

    %step 6: update state covariance
    S_prev_q1 = [eye(2,2) - K_q1*M_q1]*S_next_q1;
end
display("done");

figure()
plot(time_q1',y_q1(1:end-1),'.c')
ylabel('data (units)');
xlabel('timestamp value (sec)');
hold on;
plot(time_q1',x_q1(1:end-1,2),'.r');
title('Applying Kalman filter on a 1D constant velocity case');
legend('measurement output','filter output','location','northeast');

%%
%for a different ratio, making measurement noise more
x2_q1=zeros(length(y_q1),2);
x2_q1(1,1:end) = y_q1(1)*0.9;

x2_dot_q1 = zeros(length(y_q1),2);
x2_dot_q1(1,1:end) = y_q1(2)-y_q1(1);
```

```
var_x_dot = 0.3;
S_prev_q1 = [0, 0; 0, var_x_dot^2];
S_next_q1 = 0;
R2_q1 = 0.75;
M2_q1 = [1, 0];
Q2_q1 = [0, 0; 0, (var_x_dot^2)];

Kalman_filter(x2_q1,x2_dot_q1,y_q1,time_q1,S_prev_q1,S_next_q1,R2_q1,M2_q1,Q2_q1,2)
%%
% It can be observed that as the measurement noise increases, the filter assumes
% less confidence in sensor readings and gives more weightage to the predictions.
% Hence, it can be seen that the filter outputs are concentrated in an area while
% the measurement data is distinct at most of the places.

%for a different ratio, making dynamic noise more
x3_q1=zeros(length(y_q1),2);
x3_q1(1,1:end) = y_q1(1)*0.9;

x3_dot_q1 = zeros(length(y_q1),2);
x3_dot_q1(1,1:end) = y_q1(2)-y_q1(1);

var_x_dot = 0.3;
S_prev_q1 = [0, 0; 0, var_x_dot^2];
S_next_q1 = 0;
R3_q1 = 0.01;
M3_q1 = [1, 0];
Q3_q1 = [0, 0; 0, (var_x_dot^2)*7.5];

Kalman_filter(x3_q1,x3_dot_q1,y_q1,time_q1,S_prev_q1,S_next_q1,R3_q1,M3_q1,Q3_q1,3)
%%
% It can be observed that as the dyanmic noise increases, the filter assumes
% more confidence in sensor readings and gives less weightage to the predictions.
% Hence, it can be seen that the filter outputs are generated very similar to
% the measurement data which indicates a divergence from the predicted values.

function Kalman_filter(x2_q1,x2_dot_q1,y_q1,time_q1,S_prev_q1,S_next_q1,R_q1,M_q1,Q_q1,fign
    for i=2:1:length(y_q1)-1
        %step 1: Predict next state
        phi_q1 = [1, time_q1(i)-time_q1(i-1); 0, 1];
        X_pred_q1 = phi_q1*([x2_q1(i-1,2),x2_dot_q1(i-1,2)]');
        %placing predictions in the state variable array
        x2_q1(i,1)=X_pred_q1(1);
        x2_dot_q1(i,1)=X_pred_q1(2);

        %step 2: predict next state covariance
```

```
        S_next_q1 = phi_q1*S_prev_q1*(phi_q1') + Q_q1;

        %step 3: obtain measurements (there as input)
        Y_q1 = [y_q1(i);0];

        %step 4: calculate Kalman Gain
        K_q1 = S_next_q1*(M_q1)'*((M_q1*S_next_q1*(M_q1)'+ R_q1)^-1);

        %step 5: update state
        X_upd_q1 = X_pred_q1 + K_q1*(y_q1(i)-(M_q1*X_pred_q1));
        %placing predictions in the state variable array
        x2_q1(i,2)=X_upd_q1(1);
        x2_dot_q1(i,2)=X_upd_q1(2);

        %step 6: update state covariance
        S_prev_q1 = [eye(2,2) - K_q1*M_q1]*S_next_q1;
    end
    display("done");


    figure(fignum)
    plot(time_q1',y_q1(1:end-1),'.c')
    ylabel('data (units)');
    xlabel('timestamp value (sec)');
    hold on;
    plot(time_q1',x2_q1(1:end-1,2),'.r');
    title('Applying Kalman filter on a 1D constant velocity case');
    legend('measurement output','filter output','location','northeast');
end
```

5.2 Code: 2D Constant Velocity KF

```
clc;
clear;
urlwrite("https://cecas.clemson.edu/ahoover/ece854/labs/2D-UWB-data.txt","measurement_Q2.tx
inp_filename = "measurement_Q2.txt"
fid = fopen(inp_filename); % open the file
iCtr = 0;
yy_q2=[];
yx_q2=[];
while ~feof(fid) % loop over the following until the end of the file is reached.
  line = fgets(fid); % read in one line
  data1 = strsplit(line);
  yx_q2(end+1) = str2double(data1(1));
  yx_q2=yx_q2';
  yy_q2(end+1) = str2double(data1(2));
  yy_q2=yy_q2';
end

figure(1)
plot(yx_q2,yy_q2,'.r')
ylabel('y measurement data (units)');
xlabel('x measurement data (units)');
title('Raw output plot');
ylim([550 700])
legend('measurement output','location','northeast');


%assuming a time array corresponding with measurement data with a frequency of 1Hz
time_q2=(1:0.5:length(yy_q2)/2);
%%

%case1

%position in x
xx_q2=zeros(length(yx_q2),2);
xx_q2(1,1:end) = yx_q2(1)*0.9;
%velocity in x
xx_dot_q2 = zeros(length(yx_q2),2);
xx_dot_q2(1,1:end) = yx_q2(2)-yx_q2(1);
%position in y
xy_q2=zeros(length(yy_q2),2);
xy_q2(1,1:end) = yy_q2(1)*0.9;
%velocity in y
xy_dot_q2 = zeros(length(yy_q2),2);
xy_dot_q2(1,1:end) = yy_q2(2)-yy_q2(1);
```

```
var_xx_dot_q2 = 1;
var_xy_dot_q2 = 1;
%S_prev_q2 = [0, 0, 0, 0; 0, var_xx_dot_q2^2, 0, 0; 0, 0, 0, 0; 0, 0, 0, var_xy_dot_q2^2];
S_prev_q2 = eye(4,4);                    %4*4 matrix
S_next_q2 = 0;                           %4*4 matrix
R_q2 = [1 0.01; 0.01 1];                 %2*1 matrix
M_q2 = [1, 0, 0, 0; 0, 0, 1, 0];         %2*4 matrix
Q_q2 = [0, 0, 0, 0; 0, var_xx_dot_q2^2, 0, 0.1; 0, 0, 0, 0; 0, 0.1, 0, var_xy_dot_q2^2];

Kalman_filter(xx_q2,xx_dot_q2,xy_q2,xy_dot_q2,yx_q2,yy_q2,time_q2,S_prev_q2,S_next_q2,R_q2,
%%

%case2: decreasing measurement noise

%position in x
xx2_q2=zeros(length(yx_q2),2);
xx2_q2(1,1:end) = yx_q2(1)*0.9;
%velocity in x
xx2_dot_q2 = zeros(length(yx_q2),2);
xx2_dot_q2(1,1:end) = yx_q2(2)-yx_q2(1);
%position in y
xy2_q2=zeros(length(yy_q2),2);
xy2_q2(1,1:end) = yy_q2(1)*0.9;
%velocity in y
xy2_dot_q2 = zeros(length(yy_q2),2);
xy2_dot_q2(1,1:end) = yy_q2(2)-yy_q2(1);

vaR2_xx2_dot_q2 = 0.5;
vaR2_xy2_dot_q2 = 0.5;
%S_prev_q2 = [0, 0, 0, 0; 0, vaR2_xx2_dot_q2^2, 0, 0; 0, 0, 0, 0; 0, 0, 0, vaR2_xy2_dot_q2^
S_prev_q2 = eye(4,4);                    %4*4 matrix
S_next_q2 = 0;                           %4*4 matrix
R2_q2 = [0.1, 0.001; 0.001,0.10];     %2*1 matrix
M2_q2 = [1, 0, 0, 0; 0, 0, 1, 0];        %2*4 matrix
Q2_q2 = [0, 0, 0, 0; 0, vaR2_xx2_dot_q2^2, 0, 0.1; 0, 0, 0, 0; 0, 0.1, 0, vaR2_xy2_dot_q2^2

%Kalman_filter(xx2_q2,xx2_dot_q2,xy2_q2,xy2_dot_q2,yx_q2,yy_q2,time_q2,S_prev_q2,S_next_q2,
%%
function Kalman_filter(xx_q2,xx_dot_q2,xy_q2,xy_dot_q2,yx_q2,yy_q2,time_q2,S_prev_q2,S_next
    for i=2:1:length(yx_q2)-1
        %step 1: Predict next state
        phi_q2 = [1, time_q2(i)-time_q2(i-1), 0, 0; 0, 1, 0, 0;0, 0, 1, time_q2(i)-time_q2(
        X_pred_q2 = phi_q2*([xx_q2(i-1,2),xx_dot_q2(i-1,2),xy_q2(i-1,2),xy_dot_q2(i-1,2)]')
        %placing predictions in the state variable array
        xx_q2(i,1)=X_pred_q2(1);
```

```
        xx_dot_q2(i,1)=X_pred_q2(2);
        xy_q2(i,1)=X_pred_q2(3);
        xy_dot_q2(i,1)=X_pred_q2(4);

        %step 2: predict next state covariance
        S_next_q2 = phi_q2*S_prev_q2*(phi_q2') + Q_q2;

        %step 3: obtain measurements (there as input)
        Y_q2 = [yx_q2(i); yy_q2(i)];

        %step 4: calculate Kalman Gain
        K_q2 = S_next_q2*(M_q2)'*((M_q2*S_next_q2*(M_q2)'+ R_q2)^-1);

        %step 5: update state
        X_upd_q2 = X_pred_q2 + K_q2*(Y_q2-(M_q2*X_pred_q2));
        %placing predictions in the state variable array
        xx_q2(i,2)=X_upd_q2(1);
        xx_dot_q2(i,2)=X_upd_q2(2);
        xy_q2(i,2)=X_upd_q2(3);
        xy_dot_q2(i,2)=X_upd_q2(4);

        %step 6: update state covariance
        S_prev_q2 = [eye(4,4) - K_q2*M_q2]*S_next_q2;
    end
    display("done");


    figure(fignum)
    plot(time_q2,yx_q2(1:end-1),'.c')
    ylabel('x position (units)');
    xlabel('timestamp value (sec)');
    hold on;
    plot(time_q2,xx_q2(1:end-1,2),'.r');
    title('Applying Kalman filter on a 2D constant velocity case');
    legend('measurement output','filter output','location','northeast');
    hold off;
    figure(fignum+1)
    plot(time_q2,yy_q2(1:end-1),'.c')
    ylabel('measurement y data (units)');
    xlabel('timestamp value (sec)');
    hold on;
    plot(time_q2,xy_q2(1:end-1,2),'.r');
    title('Applying Kalman filter on a 2D constant velocity case');
    legend('measurement output','filter output','location','northeast');
end
```